IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: Craig Hansen et al. Application No.: 10/757,939 Filed: January 16, 2004 For: Multithreaded Programmable Processor and System with Partitioned Operations	Customer Number: 20277 Confirmation Number: 4645 Examiner: Jesse R. Moll Technology Center/Art Unit: 2181

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION OF CRAIG HANSEN UNDER 37 CFR § 1.131

Sir:

I, Craig Hansen, hereby declare the following to be true:

BACKGROUND

1. I am a co-inventor of United States Patent Application Number 10/757,939 (the '939 application). I understand that the '939 application is currently being reexamined by the Patent Office. Among other things, this declaration describes certain activities that occurred prior to October, 1994 through August 16, 1995, the filing date of U.S. Patent Application Serial number 08/516,036. The '036 application matured into U.S. Patent number 5,742,840, which is a parent of the '939 application. Attached to this declaration are various documents that evidence conception and diligence in reducing certain inventions claimed in the '939 application to practice.

2. I graduated from Cornell University with a Bachelor's degree in Electrical Engineering. I also received a Master's degree from Stanford University in Electrical Engineering.

3. I began my employment with MicroUnity in 1989, where I am still employed. More specifically, prior to October, 1994, and through August 16, 1995, I was employed by MicroUnity as the Chief Architect of the Terpsichore System Architecture. As such, I have personal knowledge of MicroUnity's efforts toward implementing the Terpsichore system.

4. I am currently a shareholder in MicroUnity. I own approximately five percent of the shares of the company on a fully diluted basis, plus approximately one percent additional ownership when all exercised options become vested. My current position at MicroUnity is Chief Architect.

5. All of the exhibits attached to this declaration are part of MicroUnity's records, and the exhibits are authentic and true copies of original documents (except for the redactions of dates in Exhibits A1 and A2, redactions of confidential/ privileged descriptions of the legal services provided to MicroUnity in Exhibit B, redactions of employee names and social security numbers in Exhibit E2, and the addition of exhibit and page numbers used for ease of reference).

CONCEPTION

6. Along with my co-inventor, Dr. John Moussouris, I conceived of a Multithreaded Programmable Processor and System with Partitioned Operations that included the following claims:

> 1. A programmable processor comprising:
> a data path capable of transmitting data;
> an external interface operable to receive data from an external source and communicate the received data over the data path;
> a register file containing a plurality of registers each having a register width, the register file coupled to the data path and configured to support processing of a plurality of threads and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;
> an execution unit coupled to the data path, the execution unit configured to execute a plurality of instruction streams from the plurality of threads, each instruction stream including a single instruction that specifies an arithmetic

operation to cause multiple instances of the arithmetic operation to be performed, each instance of the arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and

wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the arithmetic operation to be transmitted in parallel from the register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the arithmetic operation and execute the multiple instances of the arithmetic instruction to produce the catenated result.

2. The processor of claim 1 wherein the execution unit comprises a pipeline having a plurality of stages and wherein the pipeline interleaves execution of instructions from the plurality of instruction streams.

3. The processor of claim 2 wherein the pipeline is operable to simultaneously contain states of execution of at least two instructions from different instruction streams.

4. The processor of claim 2 wherein execution of the instructions is interleaved in a round-robin manner.

5. The processor of claim 1 wherein the processor ensures only one thread from the plurality of threads can have an exception handled at any given time.

6. The processor of claim 1 further comprising a virtual memory addressing unit and a cache operable to store data communicated between the external interface and the data path.

7. The processor of claim 1 wherein the execution unit is further operable to, in response to decoding another single instruction specifying a first and a second register each containing a plurality of floating-point operands, multiply the plurality of floating-point operands in the first register by the plurality of floating-point operands in the second register to produce a plurality of products and provide the plurality of products to partitioned fields of a result register as a second catenated result.

8. A programmable processor comprising:

a data path capable of transmitting data;

an external interface operable to receive data from an external source and communicate the received data over the data path;

first and second register files containing a plurality of registers each having a register width, the first and second register files coupled to the data path and configured to support processing of first and second threads, respectively, and to store a plurality of multiple-bit data elements in partitioned fields, each of the

multiple-bit data elements having an elemental width smaller than the register width;

an execution unit coupled to the data path, the execution unit configured to execute first and second instruction streams from the first and second threads, respectively, the first and second instruction streams each including a single instruction that specifies an arithmetic operation to cause multiple instances of the arithmetic operation to be performed, each instance of the arithmetic operation to be performed using a different one of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and

wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the arithmetic operation to be transmitted in parallel from the first register file and from the second register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the arithmetic operation and execute the multiple instances of the arithmetic instruction to produce the catenated result.

9. The processor of claim 8 wherein the execution unit comprises a pipeline having a plurality of stages and wherein the pipeline interleaves execution of instructions from the first instruction stream with instructions from the second instruction stream.

10. The processor of claim 9 wherein the pipeline is operable to simultaneously contain states of execution of an instruction from the first instruction stream and an instruction from the second instruction stream.

11. The processor of claim 9 wherein execution of the instructions is interleaved in a round-robin manner.

12. The processor of claim 9 wherein the execution unit is further operable to, in response to decoding another single instruction specifying a first and a second register each containing a plurality of floating-point operands, multiply the plurality of floating-point operands in the first register by the plurality of floating-point operands in the second register to produce a plurality of products and provide the plurality of products to partitioned fields of a result register as a second catenated result.

13. A data processing system comprising:

(a) a bus coupling components in the data processing system;

(b) an external memory coupled to the bus;

(c) a programmable microprocessor coupled to the bus and capable of operation independent of another host processor, the microprocessor comprising:

a data path capable of transmitting data;

an external interface operable to receive data from an external source and communicate the received data over the data path;

a register file containing a plurality of registers each having a register width, the register file coupled to the data path and configured to support processing of a plurality of threads and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;

an execution unit coupled to the data path, the execution unit configured to execute a plurality of instruction streams from the plurality of threads, each instruction stream including a single instruction that specifies an arithmetic operation to cause multiple instances of the arithmetic operation to be performed, each instance of the arithmetic operation to be performed using a different one of the plurality of data elements in partitioned fields of at least one of the registers to produce a catenated result; and

wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the arithmetic operation to be transmitted in parallel from the register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the arithmetic operation and execute the multiple instances of the arithmetic instruction to produce the catenated result.

14. The system of claim 13 wherein the execution unit comprises a pipeline having a plurality of stages and wherein the pipeline interleaves execution of instructions from the plurality of instruction streams.

15. The system of claim 14 wherein the pipeline is operable to simultaneously contain states of execution of at least two instructions from different instruction streams.

16. The system of claim 14 wherein execution of the instructions is interleaved in a round-robin manner.

17. The system of claim 13 wherein the processor ensures only one thread from the plurality of threads can have an exception handled at any given time.

18. The system of claim 13 further comprising a virtual memory addressing unit and a cache operable to store data communicated between the external interface and the data path.

19. The system of claim 13 wherein the execution unit is further operable to, in response to decoding another single instruction specifying a first and a second register each containing a plurality of floating-point operands, multiply the plurality of floating-point operands in the first register by the plurality of floating-point operands in the second register to produce a plurality of products and provide the plurality of products to partitioned fields of a result register as a second catenated result.

20. A data processing system comprising:

(a) a bus coupling components in the data processing system;

(b) an external memory coupled to the bus;

(c) a programmable microprocessor coupled to the bus and capable of operation independent of another host processor, the microprocessor comprising:

a data path capable of transmitting data

an external interface operable to receive data from an external source and communicate the received data over the data path;

first and second register files containing a plurality of registers each having a register width, the first and second register files coupled to the data path and configured to support processing of first and second threads, respectively, and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;

an execution unit coupled to the data path, the execution unit configured to execute first and second instruction streams from the first and second threads, respectively, the first and second instruction streams each including a single instruction that specifies an arithmetic operation to cause multiple instances of the arithmetic operation to be performed, each instance of the arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and

wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the arithmetic operation to be transmitted in parallel from the first register file and from the second register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the arithmetic operation and execute the multiple instances of the arithmetic instruction to produce the catenated result.

21. The system of claim 20 wherein the execution unit comprises a pipeline having a plurality of stages and wherein the pipeline interleaves execution of instructions from the first instruction stream with instructions from the second instruction stream.

22. The system of claim 21 wherein the pipeline is operable to simultaneously contain states of execution of an instruction from the first instruction stream and an instruction from the second instruction stream.

23. The system of claim 21 wherein execution of the instructions is interleaved in a round-robin manner.

24. The system of claim 21 wherein the execution unit is further operable to, in response to decoding another single instruction specifying a first and a second register each containing a plurality of floating-point operands, multiply the plurality of floating-point operands in the first register by the plurality of floating-point operands in the second register to produce a plurality of products and

provide the plurality of products to partitioned fields of a result register as a second catenated result.

25. The processor of claim 1 wherein the arithmetic operation comprises an integer operation.

26. The processor of claim 1 wherein the arithmetic operation comprises a floating-point operation.

27. A programmable processor comprising:

a data path capable of transmitting data;

an external interface operable to received data from an external source and communicate the received data over the data path;

a register file containing a plurality of registers each having a register width, the register file coupled to the data path and configured to support processing of a plurality of threads and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;

an execution unit coupled to the data path, the execution unit configured to execute a plurality of instruction streams from the plurality of threads, each instruction stream including a single instruction that specifies a floating-point arithmetic operation to cause multiple instances of the floating-point arithmetic operation to be performed, each instance of the floating point arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and

wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the floating-point arithmetic operation to be transmitted in parallel from the register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the floating-point arithmetic operation and execute the multiple instances of the floating-point arithmetic instruction to produce the catenated result.

28. A data processing system comprising:

(a) a bus coupling components in the data processing system;

(b) an external memory coupled to the bus;

(c) a programmable microprocessor coupled to the bus and capable of operation independent of another host processor, the microprocessor comprising:

a data path capable of transmitting data;

an external interface operable to receive data from an external source and communicate the received data over the data path;

a register file containing a plurality of registers each having a register width, the register file coupled to the data path and configured to support processing of a plurality of threads and to store a plurality of multiple-bit data elements in

partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;

an execution unit coupled to the data path, the execution unit configured to execute a plurality of instruction streams from the plurality of threads, each instruction stream including a single instruction that specifies a floating-point arithmetic operation to cause multiple instances of the floating-point arithmetic operation to be performed, each instance of the floating-point arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and

wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the floating-point arithmetic operation to be transmitted in parallel from the register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the floating-point arithmetic operation and execute the multiple instances of the floating-point arithmetic instruction to produce the catenated result.

7. It is my understanding that the claims 2-5, 9-12, 14-17 and 21-23 recited above have been rejected in the outstanding office action based on Cray, Jr. (U.S. Patent Number 4,128,880, hereinafter "Cray") in view of Chen *et al.* (U.S. Patent Number 4,771,900, hereinafter "Chen") in further view of Laudon *et al.* (U.S. Patent Number 5,938,756, hereinafter "Laudon").

8. The conception (before the publication date of Laudon, *i.e.*, before October or November, 1994) of the above-recited claims of the Multithreaded Programmable Processor and System with Partitioned Operations is shown in Exhibits A1 and A2, attached hereto. The preparation dates of cited material in Exhibits A1 and A2 are prior to October, 1994. However, the dates have been redacted in accordance with standard practice and indicated with the phrase "REDACTED".

Exhibits A1 and A2 – Presentation to Cray Research, Inc. and the pre-filing version of the Terpsichore System Architecture Manual

9. Exhibit A1 is a presentation made to Cray Research, Inc. pursuant to a confidentiality agreement between MicroUnity and Cray Research, Inc., with dates redacted. The presentation is a technology overview of the Terpsichore System Architecture and further describes certain subject matter disclosed and claimed in the '939 application. I, along with my colleagues, prepared the cited material of Exhibit A1 prior to October, 1994. Exhibit A1 contains Bates numbered pages (such as MU0020400) and all references made to Exhibit A1 will refer to the Bates numbered pages.

10. Exhibit A2 is a version of the Terpsichore System Architecture manual that I authored prior to October, 1994, with the date redacted. The manual describes certain subject matter disclosed and claimed in the '939 application. This manual was updated while work on the Terpsichore system progressed. The August 2, 1995 version of this manual was filed as an appendix to the '036 application.

11. The chart below illustrates the location and contents of the conception evidence in Exhibits A1 and A2 for each of the aforementioned claims. This chart is intended to be illustrative but not exhaustive.

'939 Claim	**Conception Evidence**
1. A programmable processor comprising:	"MicroUnity's Terpsichore System Architecture describes general-purpose processor, memory and interface subsystems . . . ." Exhibit A2, MU0023223.
1.1 **a data path capable of transmitting data;**	The data path receives data from the Non Blocking Load Control, which interfaces with external memory and other data sources such as Hermes channels, and transmits the data to the Data Cache, which in turn transmits the data over the data path to the XLU, or, alternatively, the Non Blocking Load Control transmits the data directly to the XLU. The XLU transmits the data over the data path to the Register Files and Bypass Network. The data path also connects the Register Files and Bypass

'939 Claim	Conception Evidence
	Network block to blocks such as the arithmetic logic unit (ALU), the Galois field (GF), and the local translation look-aside buffer (LTLB). Exhibit A1, MU0020439. As illustrated in the Main Pipeline block diagram, Exhibit A1, MU0020441, a 128-bit data path transmits data from the cache to the XLU, and a 64-bit data path transmits data from the MC to the cache. A 128-bit data path transmits data from the MC to the Register File (CR). Three 128-bit buses are part of the data path to transmit data from the Register File through the Bypass to the block labeled Eshort and containing the ALU. Intermediate 128-bit buses are also part of the data path to transmit data from the Eshort block, the XLU and the MC to the Bypass which, as the name suggests, bypasses the Register file.
1.2 **an external interface operable to receive data from an external source and communicate the received data over the data path;**	The Non Blocking Load Control is an external interface which receives data from an external source, such as SDRAM or a Hermes channel, and communicates the data over the data path described above. Exhibit A1, MU0020439. The Non Blocking Load Buffer provides a Processor Interface and includes a Memory buffer, a DRAM control which controls access to external DRAM memory, plus a pair of Hermes controllers which control access to Calliope interfaces. Exhibit A1, MU0020440. "The Calliope interfaces include byte-wide input and output channels intended to operate at rates of at least 1 GHz. These channels provide a packet communication link to synchronous SRAM memory on chip and a controller for interfaces to analog channels. . . . Calliope is useful in many interface applications." Exhibit A2, MU0023446.
1.3 **a register file containing a plurality of registers each having a register width, the register file coupled to the data path and configured to support processing of a plurality of threads and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-**	The Register Files and Bypass Network block indicates that it is "5x32x128", i.e., there are 5 register files each with 32 registers. Each of the registers has a register width of 128 bits. The Register Files are coupled to the data path, as shown by the arrow representing the data path and emanating from the mux at the lower right of the block diagram and extending to the Register Files

'939 Claim	Conception Evidence
bit data elements having an elemental width smaller than the register width;	and Bypass Network block. The fact that there are 5 register files configures the register files to retain data on and process 5 threads. Exhibit A1, MU0023439. "Terpsichore is designed to fetch and execute several instructions in each clock cycle. For a particular ordering o[f] instruction types, one instruction of each type may be issued in a single clock cycle. The ordering required is A, L, E, S, B; in other words, a register-to-register address calculation, a memory load, a register-to-register data calculation, a memory store, and a branch. Because of the organization of the pipeline, each of these instructions may be serially dependent." Exhibit A2, MU0023250. This instruction categorization is used to implement the processing of a plurality of threads, as depicted in Exhibit A1, MU0020364-6. The procession of the plurality of threads through the pipeline is depicted in Exhibit A1, MU0020364, in which five threads of execution are exhibited as separate pipeline stages labeled A, L, E, S, and B as above, with arrows indicating the order of operations within a single thread. There are five separate paths indicated, representing the five independent threads. The manner in which the 5 register files disclosed above support a plurality (5) of threads is shown in Exhibit A1, MU0020366. Each of the five register files, labeled "full thread 0" through "full thread 4" in the diagram, support a full thread. The "state per thread" is maintained in the register file associated with that thread. Each such register file comprises "64x64-bit registers," equivalent to the 32x128-bit registers shown in Exhibit A1, MU0023440. Moreover, each register file contains an independent "62-bit PC", or program counter, for that thread, various control bits, and a local translation lookaside buffer (LTLB), making each of the five register files essentially the equivalent for the entire register file of an unthreaded processor. Terpsichore uses fixed-point data sizes from byte (8-

'939 Claim	Conception Evidence
	bit) to hexlet (128-bit), and floating-point data sizes from half (16-bit) to quad (128-bit) precision. Exhibit A1, MU0020368-9. Terpsichore implements a number of group instructions in which a plurality of multiple-bit data elements are stored in partitioned fields and operated on by the group instruction. Each of the multiple-bit data elements has an elemental width smaller than the register width, such as the byte through octlet fixed-point sizes and the half, single and double precision floating-point sizes. Such group fixed-point instructions are depicted in Exhibit A1, MU0020371, as GMULADD8-64 and G.8-64. Group floating-point instructions are also depicted, such as GFMULADD16-64 and GF.16-64. Various group operations are shown in Exhibit A1, MU0020387-97, with a group add shown at MU0020388. In the group add, data is received from two 128-bit source registers. The data in one source register contains data elements a, b, c and d, while the other source register contains data elements e, f, g and h. In the figure, data elements a, b, c and d constitute a plurality of multiple-bit data elements stored in partitioned fields in one of the source registers and operated on by the group instruction. Each of the multiple-bit data elements a, b, c and d has an elemental width smaller than the register width. Similarly, data elements e, f, g and h in the other source register constitute a plurality of multiple-bit data elements stored in partitioned fields in a source register and operated on by the group instruction. Each of the multiple-bit data elements e, f, g and h has an elemental width smaller than the register width. As set forth in Exhibit A2, MU0023245: "The fixed-point arithmetic operations provided are most of the functions provided in the standard integer unit, except for those that check conditions. These functions include add, subtract, bitwise Boolean operations, shift, set on condition, and multiply, in forms that take packed sets of bit fields of a specified size as operands. The floating-point arithmetic operations provided are as complete as the scalar floating-point arithmetic set. The result is

'939 Claim	Conception Evidence
	generally a packed set of bit fields of the same size as the operands, except that the fixed-point multiply function intrinsically doubles the precision of the bit field. Group operations are described in more detail in Exhibit A2 at MU0023308-39. The group add instruction, as with all group instructions, is a type E instruction, i.e., a register-to-register data calculation, in this case a group add, which can be performed in any of the 5 threads in the Terpsichore architecture disclosed.
1.4 **an execution unit coupled to the data path, the execution unit configured to execute a plurality of instruction streams from the plurality of threads, each instruction stream including a single instruction that specifies an arithmetic operation to cause multiple instances of the arithmetic operation to be performed, each instance of the arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and**	The execution unit in the Terpsichore architecture includes the ALU (Exhibit A1, MU0020439), and is coupled to the data path from the block labeled Register Files and Bypass Network, as well as intermediate data paths from the LTLB, Index Adder, the block labeled GF. Using slightly different nomenclature, the execution unit in the Terpsichore architecture includes the ALU labeled Eshort at MU0020441, and is coupled to the data paths emanating from the Bypass. The execution unit is configured to execute instructions received from the instruction path from the Instruction Queue through Issue Control via the Register File. Exhibit A1, MU0020439. The manner in which the 5 Terpsichore register files support a plurality (5) of threads is shown in Exhibit A1, MU0020362. A single instruction, such as a group add, specifies an arithmetic operation such as addition and causes multiple instances of that operation to be performed, each instance of the arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result. As shown in Exhibit A1, MU0020388, the group add operation on 32-bit data adds the first data element "a" from one source register to the first data element "e" from the other source register, and places the result "a+e" in a result register. The group add instruction similarly adds the second elements of the source registers and places that result next to the first result in the

'939 Claim	Conception Evidence
	destination register, and so on until each data element in one source register is added to a corresponding data element in the other source register. The catenated result, "a+e, b+f, c+g, d+h," is placed in the destination register.
1.5 **wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the arithmetic operation to be transmitted in parallel from the register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the arithmetic operation and execute the multiple instances of the arithmetic instruction to produce the catenated result.**	In Exhibit A1, MU0020388, each data element has a width of 32 bits, and the data path is 128 bits (see, e.g., MU0020439) which is four times the data element width. This allows the data for multiple instances of the arithmetic operation to be transmitted in parallel from the register file to the execution unit. The execution unit executes the multiple instances of the arithmetic instruction in parallel to produce the catenated result, which for the group add operation is "a+e, b+f, c+g, d+h." It is apparent that the operations are mutually independent and arranged for parallel transmission and execution.
2. The processor of claim 1 wherein the execution unit comprises a pipeline having a plurality of stages and wherein the pipeline interleaves execution of instructions from the plurality of instruction streams.	See: Main Pipeline block diagram, Exhibit A1, MU0020441. The "SuperThread Pipeline" depicted in Exhibit A1, MU0020364, depicts the plurality of stages of the SuperThread pipeline, with the instructions interleaved from the plurality of instruction streams. With time moving forward going left-to-right in the diagram, each of the pipe stages A, L, E, S, B appears in a different vertical position, each associated with a distinct thread, where the threads are indicated by the arrows showing order of operations for each thread. On successive cycles, the association between the pipe stages and threads forms an interleaving pattern where each pipe stage is associated with a different thread on each cycle. While some of the pipe stages appear to be mislabeled in this diagram, the interleaving pattern is still apparent with the A, L, E, S, B stages in the first thread shown as used one cycle later for each of the remaining threads. Thus the five pipeline stages

'939 Claim	Conception Evidence
	interleave execution of instructions from the five instruction streams. The organization of the Terpsichore pipeline is discussed beginning at Exhibit A2, MU0023250: "Terpsichore is designed to fetch and execute several instructions in each clock cycle. For a particular ordering o[f] instruction types, one instruction of each type may be issued in a single clock cycle. The ordering required is A, L, E, S, B; in other words, a register-to-register address calculation, a memory load, a register-to-register data calculation, a memory store, and a branch."
3. The processor of claim 2 wherein the pipeline is operable to simultaneously contain states of execution of at least two instructions from different instruction streams.	The diagram at Exhibit A1, MU0020366, indicates that the "SuperThread state" is maintained for each of the 5 potential threads by its corresponding register file, which contains its program counter (PC), privilege level, mask bits, and local translation lookaside buffer (LTLB).
4. The processor of claim 2 wherein execution of the instructions is interleaved in a round-robin manner.	The "SuperThread Pipeline" depicted in Exhibit A1, MU0020364, depicts the plurality of stages of the SuperThread pipeline, with the instructions interleaved from the plurality of instruction streams. With time moving forward going left-to-right in the diagram, each of the pipe stages A, L, E, S, B appears in a different vertical position, each associated with a distinct thread, where the threads are indicated by the arrows showing order of operations for each thread. On successive cycles, the association between the pipe stages and threads forms an interleaving pattern where each pipe stage is associated with a different thread on each cycle. While some of the pipe stages appear to be mislabeled in this diagram, the interleaving pattern is still apparent with the A, L, E, S, B stages in the first thread shown as used one cycle later for each of the remaining threads. The interleaving pattern shows a given pipeline stage appearing for each one of the five threads on successive cycles, then returning back to these same threads, in a round-robin manner.
5. The processor of claim 1 wherein the processor ensures only one thread from the plurality of threads can have	Exhibit A1, MU0020439, entitled "Euterpe block diagram" shows exceptions which can occur from the "Address Translation" block. In Exhibit A2, MU

'939 Claim	Conception Evidence
an exception handled at any given time.	0023370 "Events and Threads", exception handling is described by "For synchronous events, Terpsichore will suspend the currently running thread in the current task, and continue execution with another thread that is dedicated to the handling of events. For asynchronous events, Terpsichore will continue execution with the dedicated event threads, while not necessarily suspending the currently running thread." This shows that there is one thread dedicated to event handling, allowing for one thread to have an exception handled at any given time.
6. The processor of claim 1 further comprising a virtual memory addressing unit and a cache operable to store data communicated between the external interface and the data path.	The "Address Translation" block in Exhibit A1, MU0020439, is the virtual memory addressing unit. Note that it is downstream of both the local translation lookaside buffer (LTLB) and global translation lookaside buffer (GTLB), confirming the fact that the Address Translation block performs virtual address translation. The "D cache/D buffer" stores data communicated between the external interface ("Non Blocking Load Control") over the data path (arrow to port "d" in the data cache).
7. The processor of claim 1 wherein the execution unit is further operable to, in response to decoding another single instruction specifying a first and a second register each containing a plurality of floating-point operands, multiply the plurality of floating-point operands in the first register by the plurality of floating-point operands in the second register to produce a plurality of products and provide the plurality of products to partitioned fields of a result register as a second catenated result.	Terpsichore uses floating-point data sizes from half (16-bit) to quad (128-bit) precision. Exhibit A1, MU0020368-9. Terpsichore implements a number of group instructions in which a plurality of floating-point data elements are stored in partitioned fields and operated on by the group instruction, and no restriction is placed on decoding a second such group floating-point instruction after a prior group floating-point instruction In Exhibit A2, MU 0023239, "Terpsichore supports several levels of precision, as well as operations to convert between these different levels. These precision levels are always powers of two, and are explicitly specified in the operation code." Such group floating-point instructions are further depicted in Exhibit A1, MU0020371, as GFMULADD16-64 and GF.16-64. Various group operations are shown in Exhibit A1, MU0020387-97, with a group add shown at MU0020388. The group add shown identifies two source registers, each of which is 128-bits. The data in one source register contains 32-bit data elements

'939 Claim	Conception Evidence
	a, b, c and d, while the other source register contains 32-bit data elements e, f, g and h. The group add operation adds the first data element "a" from one source register to the first data element "e" from the other source register, and places the result "a+e" in a result register. The group add instruction similarly adds the second elements of the source registers and places that result next to the first result in the destination register, and so on until each data element in one source register is added to a corresponding data element in the other source register. The catenated result, "a+e, b+f, c+g, d+h," is placed in the destination register. While Exhibit A1, MU0020388, illustrates a group add, the previous page, MU0020387, indicates that such group operations are performed on "Floating-point data sizes 16, 32, 64 bits" and that the operations include "Multiply." The group floating-point multiply is performed similarly to the group add instruction, except that the operation is multiply rather than add. The group floating-point multiply instructions, labeled "GF.MUL.16," "32" and "64" are further disclosed in Exhibit A2, MU0023326-8, which contain their implementation is pseudo-code.
8. A programmable processor comprising:	See claim 1.
a data path capable of transmitting data;	See claim 1.1.
an external interface operable to receive data from an external source and communicate the received data over the data path;	See claim 1.2.
first and second register files containing a plurality of registers each having a register width, the first and second register files coupled to the data path and configured to support processing of first and second threads,	See claim 1.3.

'939 Claim	Conception Evidence
respectively, and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;	
an execution unit coupled to the data path, the execution unit configured to execute first and second instruction streams from the first and second threads, respectively, the first and second instruction streams each including a single instruction that specifies an arithmetic operation to cause multiple instances of the arithmetic operation to be performed, each instance of the arithmetic operation to be performed using a different one of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and	See claim 1.4.
wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the arithmetic operation to be transmitted in parallel from the first register file and from the second register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the arithmetic operation and execute the multiple instances of the arithmetic instruction to produce the catenated result.	See claim 1.5.
9. The processor of claim 8 wherein the execution unit comprises a pipeline having a plurality of stages and	See claim 2.

'939 Claim	Conception Evidence
wherein the pipeline interleaves execution of instructions from the first instruction stream with instructions from the second instruction stream.	
10. The processor of claim 9 wherein the pipeline is operable to simultaneously contain states of execution of an instruction from the first instruction stream and an instruction from the second instruction stream.	See claim 3.
11. The processor of claim 9 wherein execution of the instructions is interleaved in a round-robin manner.	See claim 4.
12. The processor of claim 9 wherein the execution unit is further operable to, in response to decoding another single instruction specifying a first and a second register each containing a plurality of floating-point operands, multiply the plurality of floating-point operands in the first register by the plurality of floating-point operands in the second register to produce a plurality of products and provide the plurality of products to partitioned fields of a result register as a second catenated result.	See claim 7.
13. A data processing system comprising:	See claim 1.
(a) a bus coupling components in the data processing system;	Exhibit A1, MU0020439, discloses a "Peripheral Bus." MU0020440 discloses that a set of 32-bit buses connect the Non Blocking Load Buffer to DRAM and Hermes controllers.
(b) an external memory coupled to the bus;	Exhibit A1, MU0020439, discloses that the "Peripheral Bus" is connected to "SDRAM," which is a form of external memory.

'939 Claim	Conception Evidence
(c) a programmable microprocessor coupled to the bus and capable of operation independent of another host processor, the microprocessor comprising:	Exhibit A1, MU0020439, shows the processor connected to the "Peripheral Bus."
a data path capable of transmitting data;	See claim 1.1.
an external interface operable to receive data from an external source and communicate the received data over the data path;	See claim 1.2.
a register file containing a plurality of registers each having a register width, the register file coupled to the data path and configured to support processing of a plurality of threads and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;	See claim 1.3.
an execution unit coupled to the data path, the execution unit configured to execute a plurality of instruction streams from the plurality of threads, each instruction stream including a single instruction that specifies an arithmetic operation to cause multiple instances of the arithmetic operation to be performed, each instance of the arithmetic operation to be performed using a different one of the plurality of data elements in partitioned fields of at least one of the registers to produce a catenated result; and	See claim 1.4.
wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit	See claim 1.5.

'939 Claim	Conception Evidence
data elements used for the multiple instances of the arithmetic operation to be transmitted in parallel from the register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the arithmetic operation and execute the multiple instances of the arithmetic instruction to produce the catenated result.	
14. The system of claim 13 wherein the execution unit comprises a pipeline having a plurality of stages and wherein the pipeline interleaves execution of instructions from the plurality of instruction streams.	See claim 2.
15. The system of claim 14 wherein the pipeline is operable to simultaneously contain states of execution of at least two instructions from different instruction streams.	See claim 3.
16. The system of claim 14 wherein execution of the instructions is interleaved in a round-robin manner.	See claim 4.
17. The system of claim 13 wherein the processor ensures only one thread from the plurality of threads can have an exception handled at any given time.	See claim 5.
18. The system of claim 13 further comprising a virtual memory addressing unit and a cache operable to store data communicated between the external interface and the data path.	See claim 6.

'939 Claim	Conception Evidence
19. The system of claim 13 wherein the execution unit is further operable to, in response to decoding another single instruction specifying a first and a second register each containing a plurality of floating-point operands, multiply the plurality of floating-point operands in the first register by the plurality of floating-point operands in the second register to produce a plurality of products and provide the plurality of products to partitioned fields of a result register as a second catenated result.	See claim 7.
20. A data processing system comprising:	See claim 1.
(a) a bus coupling components in the data processing system;	See claim 13(a).
(b) an external memory coupled to the bus;	See claim 13(b).
(c) a programmable microprocessor coupled to the bus and capable of operation independent of another host processor, the microprocessor comprising:	See claim 13(c).
a data path capable of transmitting data	See claim 1.1.
an external interface operable to receive data from an external source and communicate the received data over the data path;	See claim 1.2.
first and second register files containing a plurality of registers each having a register width, the first and second register files coupled to the data path and configured to support processing of first and second threads,	See claim 1.3.

'939 Claim	Conception Evidence
respectively, and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;	
an execution unit coupled to the data path, the execution unit configured to execute first and second instruction streams from the first and second threads, respectively, the first and second instruction streams each including a single instruction that specifies an arithmetic operation to cause multiple instances of the arithmetic operation to be performed, each instance of the arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and	See claim 1.4.
wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the arithmetic operation to be transmitted in parallel from the first register file and from the second register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the arithmetic operation and execute the multiple instances of the arithmetic instruction to produce the catenated result.	See claim 1.5.
21. The system of claim 20 wherein the execution unit comprises a pipeline	See claim 2.

'939 Claim	Conception Evidence
having a plurality of stages and wherein the pipeline interleaves execution of instructions from the first instruction stream with instructions from the second instruction stream.	
22. The system of claim 21 wherein the pipeline is operable to simultaneously contain states of execution of an instruction from the first instruction stream and an instruction from the second instruction stream.	See claim 3.
23. The system of claim 21 wherein execution of the instructions is interleaved in a round-robin manner.	See claim 4.
24. The system of claim 21 wherein the execution unit is further operable to, in response to decoding another single instruction specifying a first and a second register each containing a plurality of floating-point operands, multiply the plurality of floating-point operands in the first register by the plurality of floating-point operands in the second register to produce a plurality of products and provide the plurality of products to partitioned fields of a result register as a second catenated result.	See claim 7.
25. The processor of claim 1 wherein the arithmetic operation comprises an integer operation.	Group fixed-point (integer) instructions are depicted in Exhibit A1, MU0020371, as GMULADD8-64 and G.8-64.
26. The processor of claim 1 wherein the arithmetic operation comprises a floating-point operation.	Group floating-point instructions are depicted in Exhibit A1, MU0020371, as GFMULADD16-64 and GF.16-64.

'939 Claim	Conception Evidence
27. A programmable processor comprising:	See claim 1.
a data path capable of transmitting data;	See claim 1.1.
an external interface operable to received data from an external source and communicate the received data over the data path;	See claim 1.2.
a register file containing a plurality of registers each having a register width, the register file coupled to the data path and configured to support processing of a plurality of threads and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the register width;	See claim 1.3.
an execution unit coupled to the data path, the execution unit configured to execute a plurality of instruction streams from the plurality of threads, each instruction stream including a single instruction that specifies a floating-point arithmetic operation to cause multiple instances of the floating-point arithmetic operation to be performed, each instance of the floating point arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and	See claim 1.4.
wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple	See claim 1.5.

'939 Claim	Conception Evidence
instances of the floating-point arithmetic operation to be transmitted in parallel from the register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the floating-point arithmetic operation and execute the multiple instances of the floating-point arithmetic instruction to produce the catenated result.	
28. A data processing system comprising	See claim 1.
(a) a bus coupling components in the data processing system;	See claim 13(a).
(b) an external memory coupled to the bus;	See claim 13(b).
(c) a programmable microprocessor coupled to the bus and capable of operation independent of another host processor, the microprocessor comprising:	See claim 13(c).
a data path capable of transmitting data;	See claim 1.1.
an external interface operable to receive data from an external source and communicate the received data over the data path;	See claim 1.2.
a register file containing a plurality of registers each having a register width, the register file coupled to the data path and configured to support processing of a plurality of threads and to store a plurality of multiple-bit data elements in partitioned fields, each of the multiple-bit data elements having an elemental width smaller than the	See claim 1.3.

'939 Claim	Conception Evidence
register width;	
an execution unit coupled to the data path, the execution unit configured to execute a plurality of instruction streams from the plurality of threads, each instruction stream including a single instruction that specifies a floating-point arithmetic operation to cause multiple instances of the floating-point arithmetic operation to be performed, each instance of the floating-point arithmetic operation to be performed using a different one of the plurality of multiple-bit data elements in partitioned fields of at least one of the registers to produce a catenated result; and	See claim 1.4.
wherein each of the multiple-bit data elements has an elemental width, and the data path has a data path width multiple times greater than the elemental width, to allow multiple-bit data elements used for the multiple instances of the floating-point arithmetic operation to be transmitted in parallel from the register file to the execution unit, and wherein the execution unit is operable to receive, in parallel, multiple-bit data elements for the multiple instances of the floating-point arithmetic operation and execute the multiple instances of the floating-point arithmetic instruction to produce the catenated result.	See claim 1.5.

12. As indicated above, Exhibits A1 and A2 were prepared before October, 1994, prior to which I spent a considerable amount of time conceiving and developing the Terpsichore System in conjunction with Dr. Moussouris. Based on the aforementioned development effort and

accompanying Exhibits A1 and A2, Dr. Moussouris and I conceived the fundamental features of the Terpsichore System prior to October, 1994, which we believed would work for their intended purpose once sufficient prototyping and testing efforts were performed.

DILIGENCE

13. Just prior to October, 1994, and through August 16, 1995, I and others at MicroUnity, as well as MicroUnity's legal patent prosecution team, were diligent in our efforts to perform detailed design, to build and to test the Terpsichore system and to file the '036 application, which eventually matured into the '840 patent, which is the original parent of the '939 application. MicroUnity's efforts to further design, build and test the Terpsichore system included the incorporation of features conceived prior to October, 1994 into integrated circuitry implementing the Terpsichore system and the development work associated with such incorporation.

14. The evidence showing diligence includes: attorney billing records from MicroUnity's patent prosecution team [Exhibit B]; e-mail communications among the MicroUnity design team making modifications to the electronic databases [Exhibit C]; modifications made to the electronic databases used to manufacture integrated circuits implementing the Terpsichore system [Exhibit D]; and MicroUnity payroll records showing salary payments to team members [Exhibit E].

Exhibit B – Attorney Billing Records

15. Exhibit B is a collection of invoices from the law firm of Willian, Brinks, Hofer, Gilson & Lione, P.C. (hereinafter "Willian Brinks"), MicroUnity's patent prosecution counsel. Descriptions of the services provided to MicroUnity have been REDACTED to maintain confidentiality/privilege. Invoice numbers appear in the upper right hand portion of the first page of each invoice. Invoice number 68811 indicates that Willian Brinks worked on the patent

application (ultimately filed as the '036 application) on April 25, 1995. Invoice numbers 70429, 72742, and 76717 illustrate that a substantial amount of work was performed by MicroUnity's patent prosecution team in preparing the patent application during the months of May, June, and July of 1995. Those efforts culminated with the filing of the '036 patent application on August 16, 1995, as indicated by the last entry on invoice number 78023.

Exhibits C6-C17 – E-mails Among the MicroUnity Design Team

16. Further evidence of MicroUnity's efforts to implement the Terpsichore system in integrated circuit form is shown in e-mail communications among the members of MicroUnity's design team from the time prior to October, 1994, through August 16, 1995. These e-mails are grouped by month and are attached hereto as Exhibits C6-C17.

17. For clarity, the chart below defines commonly used terms and acronyms related to the design phase of an integrated circuit and which, therefore, are frequently used in the e-mails contained in Exhibit C.

TERM	DEFINITION
Physical verification (or "verification")	Physical verification is a process whereby an integrated circuit layout design is checked via software tools to see if it meets certain criteria. Verification involves DRC (Design Rule Check), LVS (Layout Versus Schematic), ERC (Electrical Rule Check), XOR (Exclusive OR), and Antenna Checks.
Design Rule Checks or Checking (DRC)	DRC is a major step during physical verification of the chip design. DRC is used to determine whether a particular chip layout satisfies design rules—a series of recommended parameters provided a semiconductor manufacturer to verify the correctness of a mask set. Design Rules generally include width, spacing, enclosure, and minimum area rules. These rules exist for each layer of the semiconductor manufacturing process, with the lowest layers having the smaller rules, and the highest metal layers having larger rules. DRC software verifies that the structure meets the process constraints for a given design type and process technology. DRC software usually takes as input a layout in a standard format and a list of rules specific to the

	semiconductor process chosen for fabrication. From these it produces a report of design rule violations that the designer may or may not choose to correct.
Layout Versus Schematic (LVS)	LVS is a major step during physical verification of the chip design. A successful DRC ensures that the layout conforms to the rules designed/ required for faultless fabrication. However, it does not guarantee that the layout actually represents the circuit designers desire to fabricate. LVS checking software recognizes the drawn shapes of the layout that represent the electrical components of the circuit, as well as the connections between them. The software then compares them with the schematic or circuit diagram. When the layout matches the circuit schematic, the LVS is said to be "clean"; when they do not match, the LVS is said to be "dirty." In most cases the layout will not pass LVS the first time requiring the layout engineer to examine the LVS software's reported errors (shorts, open, component mismatch, etc.) and make changes to the layout.
Verilog	Verilog is a hardware description language (HDL) used to model electronic systems—i.e., software that can be used to program, design, and test circuits. The language (sometimes called *Verilog HDL*) supports the design, verification, and implementation of analog, digital, and mixed-signal circuits at various levels of abstraction. CSYN = a Verilog Compiler also used for logic synthesis.
SPICE (Simulation Program with Integrated Circuit Emphasis)	SPICE is a general purpose analog electronic circuit simulator. It is a powerful program that is used in IC and board-level design to check the integrity of circuit designs and to predict circuit behavior. Integrated circuits, unlike board-level designs composed of discrete parts, are impossible to breadboard before manufacture. Further, the high costs of photolithographic masks and other manufacturing prerequisites make it essential to design the circuit to be as close to perfect as possible before the integrated circuit is first built. Simulating the circuit with SPICE is the industry-standard way to verify circuit operation at the transistor level before committing to manufacturing an integrated circuit.
Tape-out (or "tapeout")	Tapeout is the final stage of the design cycle of integrated circuits or printed circuit boards, the point at which the description of a circuit is sent for manufacture. Once a chip design is complete, each layer of the chip is put on a tape for the

	integrated circuit manufacturer (or "fab") to generate a mask which then gets printed on silicon (hence the name, "tapeout"); alternately, the mask may be supplied to the fab via a file transfer program (however, this is still referred to as a tapeout).

18. Exhibit C6 is a collection of e-mail communications among the MicroUnity design team members from September 1, 1994 through September 30, 1994. As can be appreciated from inspection of Exhibit C6, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, from September through December 1994, the design team was working on major tape-out activity. (See paragraph 24, below, for a detailed tapeout schedule). Accordingly, e-mails regarding DRCs and LVS verifications, as well as CSYN compilations and related errors, for Euterpe and other related items were frequent during this month. (See *e.g.,* C6 at 10-12 [Sept 2]; 39 [Sept 7]; 47 [Sept 8]; 86 [Sept 14]; 101-104, 106, 108, 111-112, 118-119 [Sept 15]; 237 [Sept 21]; 383, 389 [Sept 24]; 426 [Sept 25]; 431-432, 434, 436-437 [Sept 26]; 487-489, 493 [Sept 28]; 595, 623, 646-647 [Sept 30]). On September 12, 1994 Jay Tomlinson outlined the necessary changes needed to implement I-cache operations, and assigned various design team members action items related to this goal (*Id.* at 66). On September 24, 1994 Lisa Robinson was "running data path tests" and discovered that "many fail." (*Id.* at 388). On September 30, 1994 Lisa Robinson checked in a number of software modifications, including changes to the swap instructions. (*Id.* at 610-622). During September, the design team also addressed routing issues and errors in Euterpe (*Id.* at 78 [Sept 13]; 334 [Sept 22]; 508. 588 [Sept 29]). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C6.

19. Exhibit C7 is a collection of e-mail communications among the MicroUnity design team

members from October 1, 1994 through October 31, 1994. As can be appreciated from inspection of Exhibit C7, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, as mentioned in paragraph 18, from September of 1994 until the end of the year, the design team was working on major tape-out activity. In an e-mail dated October 17, 1994 Paul Poenisch makes it clear that, at this point in time, both Orchis and Calliope1 were already at the fab, and the team was currently working towards the Euterpe tapeout. (C7 at 313). In an October 19, 1994 e-mail, Tim Robinson stated that "We intend to have tapes written for the base layers by Nov 1 . . .The metal layers will be at least 2 weeks later. At this point we still have substantial work to do in the SOFA to get the logic completed, debugged, routed and to make [time]." (*Id.* at 362; see also *id.* at 358-359, 361, 373 for more discussions of the Euterpe tapeout schedule). On October 21, 1994 John Mudge wrote that the "Euterpe", "Mnemo Calliope redo" and "Pollux redo" tapeouts were all upcoming at that time, and that, "[u]sing a rough figure of 3 months to complete a tapeout . . . [t]hen we have, now 31-Dec for euterpe, Jan-31 Mar for Mnemo." (*Id.* at 477-479). Accordingly, e-mails regarding DRCs and LVS verifications, as well as CSYN compilations and related errors, for Euterpe and other related items were plentiful and frequent during this month. (See *e.g.,* C7 at 22 [Oct 2]; 27 [Oct 3]; 208 [Oct 11]; 217-220 [Oct 12]; 275 [Oct 15]; 415-416, 428 [Oct 20]; 483-484 [Oct 21]; 513-517 [Oct 23]; 569 [Oct 26]; 598 [Oct 27]). On Oct 22, 1994 Lisa Robinson announced that, after numerous attempts and error corrections throughout the day (see *id.* at 489-507), "I believe that the [Euterpe] build is complete" (*Id.* at 510), and both Tom Laidig and Mark Hoffman responded with "Congratulations!" (*Id.* at 511-512). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C7.

20. Exhibit C8 is a collection of e-mail communications among the MicroUnity design team members from November 1, 1994 through November 30, 1994. As can be appreciated from inspection of Exhibit C8, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, as detailed in paragraphs 18-19 above, from September of 1994 until the end of the year, the design team was focused on major tape-out activity. Accordingly, e-mails regarding DRCs and LVS verifications, as well as CSYN compilations and related errors, were plentiful and frequent during this month. (See *e.g.,* C8 at 24-26 [Nov 2]; 116-118 [Nov 3]; 121-122, 133 [Nov 4]; 147-148, 163-164, 169, 170 [Nov 5]; 224-225 [Nov 6]; 238-239, 242-243 [Nov 7]; 256, 261-265 [Nov 8]; 273 [Nov 9]; 300, 360 [Nov 11]; 362 [Nov 12]; 390 [Nov 14]; 563 [Nov 28]). On November 10, 1994 Graham Y. Mostyn circulated an e-mail stating "[n]ow that the main board layout is close to completion, we would like all of the circuit designers to check the final captured schematics and this interim layout, in particular tracing out the sections that they own. We're therefore circulating tomorrow (Friday) schematics and plots to the following people, and would appreciate it if you could report to Arya at or before the 3PM Monday netlist meeting of discrepancies in any of the three databases." (*Id.* at 316; see also *id.* at 365). On November 11 and 12, 1994 e-mails with the subject line "hestia power planes" began circulating regarding the finalization of the printed circuit board (PCB) layout and stack-up for the system. (*Id.* at 343, 345, 355, 357, 386). Between November 14 and 15, 1994 members of the design team also spent time debating the need for a Euterpe perforation treatment, and potential ways to implement such a treatment. (See *id.* at 390, 395, 412-413, 423, 427). Time was also spent developing and implementing a realtime software test plan for Euterpe and Calliope. (See *id.* at 495-497). Numerous additional implementation details of the Terpsichore system during this time period

are evident from inspection of Exhibit C8.

21. Exhibit C9 is a collection of e-mail communications among the MicroUnity design team members from December 1, 1994 through December 31, 1994. As can be appreciated from inspection of Exhibit C9, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, as detailed in paragraphs 18-20 above, from September of 1994 until the end of the year, the design team was focused on major tape-out activity. Accordingly, e-mails regarding DRCs and LVS verifications, as well as CSYN compilations and related errors, were plentiful and frequent during this month. (See *e.g.* C9 at 38 [Dec 3]; 46 [Dec 5]; 141-144 [Dec 6]; 167-168 [Dec 7]; 180-181, 186, 191-193 [Dec 8]; 442, 452-460 [Dec 19]; 472 [Dec 21]; 484 [Dec 23]; 560-564 [Dec 31]). On December 1, 1994 the final printed circuit board (PCB) meeting prior to release to fab was held. (*Id.* at 5). Tom Eich sent out a summary of the meeting and an action list of items to be completed before the PCB was sent to fab. (*Id.*). "Software Bringup Meetings" were held on December 7, 14, and 21, 1994 covering general software development for the system and including specific action items to be completed each week by design personnel. (*Id.* at 155 [Dec 7]; 294 [Dec 14]; 467 [Dec 21]). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C9.

22. Exhibit C10 is a collection of e-mail communications among the MicroUnity design team members from January 1, 1995 through January 31, 1995. As can be appreciated from inspection of Exhibit C10, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, e-mails regarding DRCs and LVS verifications, as well as CSYN compilation attempts and errors, continued to be plentiful during this month. (See *e.g.*, C10 at 7 [Jan 3]; 99 [Jan 9]; 120 [Jan 11]; 150 [Jan 12];

217 [Jan 18]; 222 [Jan 19]; 263 [Jan 22]; 265 [Jan 23]; 280 [Jan 24]; 311 [Jan 28]; 330 [Jan 30]). Design and implementation of specific group instructions were also discussed—for example, numerous e-mails circulated discussing the implementation of the GEXTRACT.128 instruction. (C10 at 43 [Jan 5], 73-74 [Jan 6], and 81 [Jan 7]). On January 31, 1995, Tim Robinson sent an e-mail requesting a meeting to "formalize the timeline for Euterpe tapeout." (C10 at 345). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C10.

23. Exhibit C11 is a collection of e-mail communications among the MicroUnity design team members from February 1, 1995 through February 28, 1995. As can be appreciated from inspection of Exhibit C11, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, throughout the month design rule conformance was continuing to be verified (C11 at 19 [Feb 2]; 72 [Feb 3]; 99 [Feb 5]; 130 [Feb 7]; 159 [Feb 8]; 194 [Feb 9]; 226 [Feb 11]; 251 [Feb 12]; 299-300 [Feb 14]; 454 [Feb 22]; 548 [Feb 28]), and CSYN compilation attempts and correction of associated errors continued. (C11 at 153 [Feb 8]; 245 [Feb 11]; 263 [Feb 13]; 340 [Feb 17]; 405 [Feb 21]; 559 [Feb 28]). Also, the e-mails reflect continuing debugging of the design, such as cache functionality and tagging (C11 at 56 [Feb 3]; 182 [Feb 9]) and GARDS (gate array design system) design development (C11 at 268 [Feb 13]; 327 [Feb 16]; 391 [Feb 20]; 486 [Feb 25]; 505 [Feb 26]). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C11.

24. Exhibit C12 is a collection of e-mail communications among the MicroUnity design team members from March 1, 1995 through March 31, 1995. As can be appreciated from inspection of Exhibit C12, during this time period the design team was continually working to implement

the Terpsichore system in integrated circuit form. For example, throughout the month design rule conformance was continuing to be verified (C12 at 28 [Mar 2]; 231 [Mar 14]; 281 [Mar 15]; 343-45 [Mar 17]; 507 [Mar 27]; 617 [Mar 31]), and CSYN compilation attempts and correction of associated errors continued. (C12 at 1 [Mar 1]; 201 [Mar 11]; 278 [Mar 15]; 337 [Mar 17]). In March 1995, details of the Euterpe design were continuing to be refined and corrected. (C12 at 31 [Mar 2]; 236 [Mar 14]; 290 [Mar 16]; 343 [Mar 17]; 501 [Mar 26]; 626 [Mar 31]). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C11.

25. Exhibit C13 is a collection of e-mail communications among the MicroUnity design team members from April 1, 1995 through April 30, 1995. As can be appreciated from inspection of Exhibit C13, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, on April 5, 1995 Dave Van't Hof communicated the results of a layout-versus-schematic (LVS), a tool commonly used to ensure that the electronic databases indeed represent the schematics the designers intended. Mr. Van't Hof indicated that the LVS results were the "best results we've had in a long time." (Exhibit C13 at 66). Furthermore, on Friday, April 21, 1995 Geert Rosseel discussed designing the Terpsichore system such that it could be manufactured at multiple foundries including Taiwan Semiconductor Manufacturing Corporation (TSMC). (Exhibit C13 at 30). Also, on Sunday, April 1, 1995 Tom Eich and Tim Robinson discussed the mechanical details of the Hermes modules, and their impact on printed circuit board layouts of other modules. (Exhibit C13 at 1). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C13.

26. Exhibit C14 is a collection of e-mail communications among the MicroUnity design team members from May 1, 1995 through May 31, 1995. As can be appreciated from inspection of Exhibit C14, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, on May 3, 1995 a meeting was held discussing the status of several chips being manufactured, including the Euterpe block (discussed above in paragraph 10) and the Calliope block (discussed above in paragraph 10). (*Id.* at 151). The status and forecasting for Euterpe were further discussed on May 28, 1995. (*Id.* at 22). Further, as of May 31, 1995 Calliope was being processed through the fab and Paul Poenisch informed the MicroUnity design team that new reticles (which are discussed below in paragraph 35) for Calliope were needed (Exhibit C14 at 9). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C14.

27. Exhibit C15 is a collection of e-mail communications among the MicroUnity design team members from June 1, 1995 through June 30, 1995. As can be appreciated from inspection of Exhibit C15, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, on Friday, June 9, 1995 Dave Van't Hof checked transistors within Euterpe to determine if proposed new design rules from the fab would require Euterpe to be significantly modified. (Exhibit C15 at 80). Design rules may be thought of as tolerances that integrated circuits manufactured in the fab should conform to in order to obtain a certain level of functional circuits from the fab. These design rules are checked using design-rule-checking (DRC) software. On Tuesday, June 20, 1995 Geert Rosseel requested that the latest design rules be sent to the members of the design team so that Euterpe could be checked for compliance. (*Id.* at 39). Also on Tuesday, June 20, 1995, Tim Robinson

discussed testing Euterpe. (*Id.* at 27). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C15.

28. Exhibit C16 is a collection of e-mail communications among the MicroUnity design team members from July 1, 1995 through July 31, 1995. As can be appreciated from inspection of Exhibit C16, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, on Sunday, July 9, 1995 Tim Robinson sent an e-mail regarding the status of Euterpe's physical design and its timing estimates. (Exhibit C16 at 94). In addition, on Saturday, July 29, 1995 Tim Robinson announced that Euterpe had completed testing at 1GHz with no errors. (*Id.* at 20). Also, on Monday, July 31, 1995 a meeting was held to discuss the status of Euterpe and goals to produce a Euterpe design consistent with the current design rules. (*Id.* at 7). Numerous additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C16.

29. Exhibit C17 is a collection of e-mail communications among the MicroUnity design team members from August 1, 1995 through August 16, 1995. As can be appreciated from inspection of Exhibit C17, during this time period the design team was continually working to implement the Terpsichore system in integrated circuit form. For example, on Sunday, August 6, 1995 Dave Van't Hof announced that an LVS of the full chip Euterpe had finished. (Exhibit C17 at 117). Also, on Monday, August 7, 1995 a meeting was held where Euterpe was discussed including its ability to complete "between 0.3 and 0.4 instructions/cycle in a single thread." (*Id.* at 108). Further, on Tuesday, August 15, 1995 a review of Euterpe's electronic database or "layout" was conducted where items requiring fixing were identified (*Id.* at 6). Numerous

additional implementation details of the Terpsichore system during this time period are evident from inspection of Exhibit C17.

Exhibits D1 and D23-D70 – Logs of Modifications to the Electronic Databases

30. MicroUnity's Terpsichore system was intended to be implemented in integrated circuit form. Semiconductor integrated circuits are typically fabricated in a layer process which includes the key process steps of imaging, deposition, and etching. These main process steps may be supplemented by doping, cleaning and planarization steps.

31. Integrated circuit fabrication facilities, often called "fabs", manufacture integrated circuits on wafers of semiconducting material—generally silicon—referred to as the substrate. Photolithography is used to mark different areas of the substrate to be doped or to have polysilicon, insulators or metal (typically aluminum) tracks deposited on them. Integrated circuits are composed of many overlapping layers, each defined by photolithography, that are normally represented by different colors in software simulation tools. Some layers mark where various dopants are diffused into the substrate (called diffusion layers), some define where additional ions are implanted (implant layers), some define the conductors (polysilicon or metal layers), and some define the connections between the conducting layers (via or contact layers). All components are constructed from a specific combination of these layers.

32. The substrate is marked via the use of various reticles (also referred to as "photoreticles," "masks," and "photomasks"), opaque plates with holes or transparencies that allow light to shine through in a defined pattern. Accordingly, the location of the circuitry on these wafers, and ultimately the function of the circuitry, is determined by a series of steps that include shining light through a number of reticles to generate the desired circuitry.

33. Reticles are produced using very elaborate electronic databases that typically involve a great deal of time and money to produce. Sometimes, these electronic databases are also referred to as "tapeouts" or "physical layouts" of the integrated circuits in question. The electronic databases also contain layouts of individual cells which are combined to produce the final layout, as well as schematic representations of the circuits contained in the cells, Verilog or logic-equation representations of the design, documentation of the design, and software executed on the processor. Of particular interest are these electronic databases that represent the results of personal human work most directly, as the final resulting "tapeout" is formed using elaborate computer programs operating on copies of the electronic databases. The common electronic database utilized at MicroUnity is a CVS system, where CVS stands for Concurrent Versions System; Concurrent because many users may concurrently check-out and modify copies of the database, each developing their own version, which may then be checked-in to the database. Records of modifications made to these databases are contained in a CVS log. Whenever a single person, or small group of people, made, tested, and committed to making a modification to the database, there is a corresponding "check-in" to the database, reflecting that such a modification was made. It should be noted, however, that the number of modifications [i.e., CVS log entries] does not reflect directly on the character of the changes made—whether a file changes just a little, or changes a lot, with a great many individual changes all lumped together, it is still logged as a single modification to the database.

Modifications to the electronic database indicate that design work on a particular integrated circuit layout is still being done. As a design group gets closer to completing a chip design, the level of revisions quickly goes down to stop any further design work—the design is effectively "locked down" for an upcoming tapeout, and designers focus on fixing errors

(encountered, for example, as a result of Design Rule Checking (DRC) and Layout Versus Schematic (LVS) procedures to test and prepare the chip for tapeout) rather than making design changes. Accordingly, times where there are relatively few entries in the CVS log correspond to preparation for tapeout activity for MicroUnity's chip designs. E-mails contained in Exhibit C support this by documenting work towards tapeout during and just prior to these periods (see specifically, e-mails discussing DRC, LVS, and other such activities).

34. The Terpsichore system design team included over 30 individuals at all relevant times (*see* E1, below). In my capacity as Chief Architect, I was kept abreast of the design team's efforts to implement the Terpsichore system. From a time prior to October, 1994 through August 16, 1995, the individuals on this design team spent a substantial amount of their time in building elaborate common databases (as described above) for the Terpsichore system. Exhibits D2-D70 represent weekly logs of modifications to the electronic databases for the Terpsichore system from September 20, 1994 through August 16, 1995. Exhibit D1 is a summary, on a weekly basis, of the number of modifications made to these electronic common databases during this time period. All told, there were approximately 14,859 changes to the Terpsichore system electronic databases from September 20, 1994 through August 16, 1995. Below is a chart indicating the correlation between usernames for "authors" in the CVS logs, and the corresponding MicroUnity employees.

CVS username	**MicroUnity Employee**
ONG	Warren Ong
Tbr	Tim Robinson
Tom	Tom Eich
Hopper	Mark Hofmann
AGC	Alan Corry
FWO	Fred Obermeier
Bobm	Bob Morgan

Wampler	Kurt Wampler
Brian	Brian Smith
Brianl	Brian Lee
Claseman	Tim Claseman
Yves	Jean-Yves Michel
Chuck	Chuck Rasbold
Bruce	Bruce Baternan
Jeff	Jeff Marr
LisaR	Lisa Robinson
Woody	Jay Tomlinson
VO	Tom Vo
DOI	Derek Iverson
Veena	Veena Malwankar
Guarino	Lorreta Guarino-Reid

In addition to the electronic common databases, members of the design team generally checked-out and maintained individual copies of the common databases for the purpose of generating modifications and additions to the databases, and for the purposes of testing the design or portions of the design. These individual copies of the common databases were modified at will, and such changes were not logged in the common database until the modified copies were then checked-in. As a general rule, members of the design team would perform individual testing of these modifications and additions prior to checking them into the electronic common databases. When checking in these modifications, the design team member would be expected to make a note in the common databases regarding the nature or purpose of the change or addition. Often, more than one file was checked-in at once, and the single note supplied by the design team member is repeated for each modified file in the database.

35. Exhibits D23-D70 show modifications on a weekly basis, with entries showing the file modified in the database, the user name of the team member, the note entered in the database, and the number of text lines added or removed. Because numerous files may be checked in all at once, many entries will show the same note for each file. The notation "initial revision" is

commonly used to remark that this check-in is the first time that a file has been checked into one of the common databases.

Weekly periods during which few or no modifications have been entered into the common databases can occur. One reason for such occurrences is that during these periods, the design team may be concentrating on testing or integration of the design. Executing the final steps of a design requires extensive testing and long compilation times, as the Verilog database is compiled into a gate-level design and the gate-level design is placed on the chip and the wires are routed between gates. Logic verification, timing analysis, and layout-vs-schematic checking are all tasks that are performed near the "tape-out" time, and on the computers of the time, often consumed several consecutive days of computer time for each task. Another reason for such occurrences is that team members performed much of their design work on individual databases prior to checking in modifications to the common databases. Thus this design work is reflected on the common databases later, at the time the design work is checked in.

Layout databases were managed by design team members known as "Mask Designers," that did not directly interact with the electronic common databases. However, their work was periodically inserted into the electronic common databases, usually with a dummy user name "chip" with a note indicating a periodic check-in.

36. Exhibit D23 shows the modifications made from September 20-26, 1994. These modifications included approximately 6 changes to the Terpsichore system. (Exhibit D1). For example, on September 20, 1994 Lorreta Guarino-Reid checked in a modification to "use tcc, not tscc" noted that after other revisions he "expect[ed] a few floating-point errors" to occur. On September 21, 1994 Lorreta Guarino-Reid "[made] expected results consistent with one another." Other modifications were made to the Terpsichore system's electronic database during

this week as shown in Exhibit D23. Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibit C6 (discussed in paragraph 18, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

37. Exhibit D24 shows that there were no modifications checked in to the electronic database from September 27-October 3, 1994. (Exhibit D1, D24). Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibits C6-C7 (discussed in paragraphs 18-19, above) contain numerous e-mails detailing the continuous design and implementation activity during this time period.

38. Exhibit D25 shows that there were no modifications checked in to the electronic database from October 4-10, 1994. (Exhibit D1, D25). Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibit C7 (discussed in paragraph 19, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

39. Exhibit D26 shows the modifications made from October 11-17, 1994. There was one change made to the Terpsichore system this week—Jean-Yves Michel made an "initial checkin" of a file. (Exhibit D1; D26 at 1). Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibit C7 (discussed in paragraph 19, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

40. Exhibit D27 shows the modifications made from October 18-24, 1994. These modifications included approximately 7 changes to the Terpsichore system. (Exhibit D1). For

example, on October 20, 1994 Tim Claseman "add[ed] debug information" a "periodic checkin" was made of layout files that had been worked on. (D27 at 1). Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibit C7 (discussed in paragraph 19, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

41. Exhibit D28 shows the modifications made from October 25-31, 1994. These modifications included approximately 14 changes to the Terpsichore system. (Exhibit D1). For example, on October 25, 1994 Tim Claseman "execute[d] file cmd on obj". (D28 at 2). On October 26, 1994 he "correct[ed] file logic error, and on October 28, 1994 he checked in a "file type after compiler before link" modification and also "correct[ed the] compiler path." (*Id.* at 1). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D28.

42. Exhibit D29 shows that there were no modifications checked in to the electronic database from November 1-7, 1994. (Exhibit D1, D29). Paragraph 40 (above) explains why some require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibit C8 (discussed in paragraph 20, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

43. Exhibit D30 shows the modifications made from November 8-14, 1994. These modifications included approximately 22 changes to the Terpsichore system. (Exhibit D1). For example, on November 10, 1994 Alan Corry added "additional scaling modes for accumulator" and "support for additional accumulator scaling modes." (D30 at 1, 6). He also "adjust[ed] output packing to accept data left adjusted rather than right adjusted." (*Id.* at 6). Numerous other

modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D30.

44. Exhibit D31 shows the modifications made from November 15-21, 1994. These modifications included approximately 5 changes to the Terpsichore system. (Exhibit D1). For example, on November 16, 1994 Tim Claseman "add[ed] additional tests to Plum Hall run" (D31 at 1). On November 18, 1994 Tim Claseman "correct[ed] execution of testing directory" and "added exin tests to Plum Hall run." (*Id.* at 1). Other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D31. Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibit C8 (discussed in paragraph 20, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

45. Exhibit D32 shows that there were no modifications checked in to the electronic database from November 22-28, 1994. (Exhibit D1, D32). Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibit C8 (discussed in paragraph 20, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

46. Exhibit D33 shows the modifications made from November 29-December 5, 1994. These modifications included approximately 4 changes to the Terpsichore system. (Exhibit D1). For example, on November 30, 1994 Tim Claseman "update[d] conform.exp." (D33 at 1). On December 1, 1994 Chuck Rasbold "changed make variable 'LANG'to be 'LANG_' instead," explaining that "the name 'LANG' is a poor choice since gmake puts the make variables into the environment, and many implementations of setlocale() will read the environment variable

'LANG.'" (*Id.* at 1). On December 2, 1994 Alan Corry checked in "scaling mode inputs" and corrected a "missing variable declaration." (*Id.* at 1-2). Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibits C8-C9 (discussed in paragraphs 20-21, above) contain numerous e-mails detailing the continuous design and implementation activity during this time period.

47. Exhibit D34 shows the modifications made from December 6-12, 1994. There was one change made to the Terpsichore system this week—Fred Obermeier "define[d] CHIP macro for use by celltest and related binaries." (Exhibit D1; *Id.* at 1). Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibit C9 (discussed in paragraph 21, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

48. Exhibit D35 shows the modifications made from December 13-20, 1994. These modifications included approximately 55 changes to the Terpsichore system. (Exhibit D1). For example, on December 13, 1994 Tim Claseman "[made] expected results consistent with one another." Throughout the rest of the week he continued to make and check-in "initial revisions." (See, generally, D35; see also paragraphs 39-40, above, discussing initial revisions). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D35.

49. Exhibit D36 shows the modifications made from December 20-26, 1994. These modifications included approximately 32 changes to the Terpsichore system. (Exhibit D1). During this week, Tim Claseman made and checked-in numerous "initial revisions." (See,

generally, D36; see also paragraphs 39-40, above, discussing initial revisions). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D36.

50. Exhibit D37 shows the modifications made from December 27, 1994 -January 2, 1995. These modifications included approximately 6 changes to the Terpsichore system. (Exhibit D1). For example, on December 28, 1994 Brian Smith "added some new tests to run an impulse through all of co." (D37 at 1). Paragraph 40 (above) explains why some periods require fewer modifications to the electronic database despite ongoing diligent efforts toward reduction to practice. Exhibits C9-C10 (discussed in paragraphs 21-22, above) contain numerous e-mails detailing the continuous design and implementation activity during this time period.

51. Exhibit D38 shows the modifications made from January 3-9, 1995. These modifications included approximately 4 changes to the Terpsichore system. (Exhibit D1). For example, on January 6, 1995 Brian Smith "added a note about scale factors used throughout co," "added an impulse test," and "turned on some debug output in sim-mixer.v." (D38 at 1). Paragraph 40 (above) explains why some periods require fewer or no modifications to the electronic database despite ongoing diligent efforts towards reduction to practice. Exhibit C10 (discussed in paragraph 22, above) contains numerous e-mails detailing the continuous design and implementation activity during this time period.

52. Exhibit D39 shows the modifications made from January 10-16, 1995. These modifications included approximately 1800 changes to the Terpsichore system. (Exhibit D1). For example, on January 14, 1995, Tom Eich "fix[ed] atlas Makefiles to build from top to bottom" and released "initial atlas, with minimal ged directory." (D39 at 1; see also paragraph 40, above, discussing "initial revisions"). He also check-in files "copied from proteus." (*Id.* at

24). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D39.

53. Exhibit D40 shows the modifications made from January 17-23, 1995. These modifications included approximately 252 changes to the Terpsichore system. (Exhibit D1). For example, on January 17, 1995 Drew Wingard "fix[ed] RCS version info string," and on January 18, 1995 he "improve[d] doc.h macros." On January 19, 1995 Geert Rosseel "added [a] clean-mechanism," "added .html rules," and checked in an "initial release." (D40 at 1). On January 21, 1995 Warren Ong performed an "initial check in of [the] stdcell template schematic" and noted "new bodies added or bodies modified." (*Id.* at 7; see also paragraphs 39-40, above, discussing initial revisions). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D40.

54. Exhibit D41 shows the modifications made from January 24-30, 1995. These modifications included approximately 12 changes to the Terpsichore system. (Exhibit D1). For example, on January 26, 1995 Geert Rosseel checked in "register file documentation" to several files and added to the BOM (Bill of Materials) file. (D41 at 1; see also paragraphs 39-40, above). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D41.

55. Exhibit D42 shows the modifications made from January 31-February 6, 1995. These modifications included approximately 144 changes to the Terpsichore system. (Exhibit D1). For example, on February 1, 1995 Geert Rosseel checked in "CSM solder design rules" (D42 at 7) and Warren Ong made "misc[ellaneous] edits mostly to location properties." (*Id.* at 22). On February 2, 1995 Geert Rosseel "released[d] solder design data rule" and "release[d] layout.html." (*Id.* at 4). On February 3, 1995 Geert Rosseel performed a "cache initial check-in"

and "added physical constraints," and on February 4, 1995, he "added . . . rules for custom block domensions [*sic*]." (*Id.* at 5-7). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D42.

56. Exhibit D43 shows the modifications made from February 7-13, 1995. These modifications included approximately 519 changes to the Terpsichore system. (Exhibit D1). For example, on February 7, 1995 Tom Eich made a "first cut at atlas compass area." (D43 at 23). On February 9, 1995 Warren Ong "fixed beta parameter on body," "removed 3rd and 4th input pins," and "changed [the] ordering of inputs s.t. msb=most slowest bit." (D43 at 31). On February 11, 1995 Brian Lee "added rules for xl schematic generation" and "added parmsche variable." (*Id.* at 1). On February 12, 1995 Geert Rosseel "built a baseplate with a pad-ring and holes for the custom blocks." (*Id.* at 3). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D43.

57. Exhibit D44 shows the modifications made from February 14-20, 1995. These modifications included approximately 630 changes to the Terpsichore system. (Exhibit D1). For example, on February 15, 1995 Brian Lee "add[ed] logic mapping page." (D44 at 72). On February 17, 1995 Brian Smith "add[ed] muxes" and Warren Ong "added series feedback PMOS" and added "new schematics based on xvmux8n." (D44 at 1, 33). Several files containing layout databases had "periodic checkin" of modified layout data performed (*Id.* at 4-24). The mechanism by which this occurs is described in paragraphs 39-40, above. Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D44.

58. Exhibit D45 shows the modifications made from February 21-27, 1995. These modifications included approximately 5222 changes to the Terpsichore system. (Exhibit D1).

For example, on February 22, 1995 Kurt Wampler checked in an "initial release of atlas/gards Makefiles & basegenstuff" as well as an "initial release of atlas/gards/basegen scripts/programs." (D45 at 1). On the same day, Warren Ong "size[d] transistors" as "discharge transistor is a different size than the logic transistors," and checked in "new schematics based on the 17-input schematic, with sizes." (*Id.* at 104-105, 227-228, 231-250). On February 24, 1995 Kurt Wampler "fixed [a] bug in sofacdl with regard to recognition of cell profile identifiers," (*Id.* at 1) and Brian Lee "added dand* dor* cells." (*Id.* at 126). Several files containing layout databases had "periodic checkin" of modified layout data performed (*Id.* at 2-90, 130-216). The mechanism by which this occurs is described in paragraphs 39-40, above. On February 24, 1995 Tim Claseman checked in an "initial revision" of the "Nullstone" compiler test suite, and then added the notation of "Nullstone version 3.8.4" for numerous individual files in the compiler test suite. (*Id.* at 253-945). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D45.

59. Exhibit D46 shows the modifications made from February 28-March 6, 1995. These modifications included approximately 1531 changes to the Terpsichore system. (Exhibit D1). For example, on February 28, 1995 Tim Claseman checked in "plumhall update version 6.0," another compiler test suite with numerous files. (D46 at 136-213). On March 3, 1995, Warren Ong checked in "new parmsche parameters on bodies" (*Id.* at 38) and "changed parmsche parameters from 'NMOS' to 'NSIZE' and 'PMOS' to 'PSIZE.'" (*Id.* at 39). On the same day, Brian Lee "fix[ed] power level extraction." (*Id.* at 67). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D46.

60. Exhibit D47 shows the modifications made from March 7-13, 1995. These modifications included approximately 386 changes to the Terpsichore system. (Exhibit D1). For example, on

March 7, 1995 Warren Ong made modifications so that "all xv bodies now have size parameter specifications" (D47 at 1) and "change[d the] formula for using the fanout parameter" (*Id.* at 56), and Geert Rosseel "made a bunch of changes" including "add[ing] a better model for the clock-tree" and including "better pads." (*Id.* at 2). Numerous files containing layout databases had "periodic checkin" of modified layout data performed (*Id.* at 3-54). The mechanism by which this occurs is described in paragraphs 39-40, above. Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D47.

61. Exhibit D48 shows the modifications made from March 14-20, 1995. These modifications included approximately 270 changes to the Terpsichore system. (Exhibit D1). For example, on March 14, 1995 Tom Eich "change[d] more schematic names to 'xv<fanin><function><polarity>' format." (D48 at 19). On March 15, 1995 Geert Rosseel made "more baseplate changes . . . added hemming cells [and] added in real pads." Numerous files containing layout databases had "periodic checkin" of modified layout data performed (*Id.* at 2-14). The mechanism by which this occurs is described in paragraphs 39-40 above. Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D48.

62. Exhibit D49 shows the modifications made from March 21-27, 1995. These modifications included approximately 755 changes to the Terpsichore system. (Exhibit D1). For example, on March 22, 1995 Kurt Wampler checked in the following modifications: "basecompile: fixed global pin handling[;] flat.sel: added power pads etc.[;] sofa_model.ly.template: updated locations of 3 global pins [-] vss, vdd, phi_am." (D49 at 32). On March 23, 1995 Geert Rosseel checked in "nb buffer specifications." (*Id.* at 83). On March 24, 1995 Tom Eich "added rdiff, rdiffs, rnw, and spnp." (*Id.* at 1). Numerous other modifications

were made to the Terpsichore system's electronic database during this week as shown in Exhibit D49.

63. Exhibit D50 shows the modifications made from March 28-March 31, 1995. These modifications included approximately 1108 changes to the Terpsichore system. (Exhibit D1). For example, on March 29, 1995 Tom Eich made a "first shot at building xs cells" and noted that it "only works for hierarchical cells, so far." (D50 at 4). On the same day, Tom Eich "added strength parameters to composite cells" and "changed brackets to braces in some bodies." (*Id.* at 50). On March 31, 1995 Warren Ong made "edits for ged2spice compilation" (*Id.* at 3) and "added new schematics for generating xs* cells." (*Id.* at 73). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D50.

64. Exhibit D51 shows the modifications for the week beginning on April 1, 1995. These modifications included approximately 153 changes to the Terpsichore system. (Exhibit D1). For example, on April 3, 1995 Richard Dickson "moved pc to bit positions 63:0 from 127:64" (D51 at 3) and on April 4, 1995 Bill Zuravleff "added interface signal." (*Id.* at 1). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D51.

65. Exhibit D52 shows the modifications for the week beginning on April 8, 1995. These modifications included approximately 184 changes to the Terpsichore system. (Exhibit D1). For example, on April 9, 1995 Mark Semmelmeyer "add[ed] 5 stage shift register & snapper so ICC can capture a gva[63:48] on branches and thus no longer need to contin[u]ous ltlb of its own and so it can get high gva in time for use on target" (D52 at 16) and on April 12, 1995 Mark Semmelmeyer "change[d] reset PC from 0000_4000_0000_0000 to 0000_4000_0000_0200."

(*Id.* at 2). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D52.

66. Exhibit D53 shows the modifications for the week beginning on April 15, 1995. These modifications included approximately 148 changes to the Terpsichore system. (Exhibit D1). For example, on April 16, 1995 Richard Dickson "add[ed] components for buffer copy of address output" (D53 at 36) and on April 20, 1995 Bill Zuravleff "[u]pdated placement to fix NBhc1prbgrant timing path and cells flopping over clock spar on rhs." (*Id.* at 1). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D53.

67. Exhibit D54 shows the modifications for the week beginning on April 22, 1995. These modifications included approximately 209 changes to the Terpsichore system. (Exhibit D1). For example, on April 24, 1995 Mark Semmelmeyer "[s]hortened snake read snap by 1 tick to match hr-->ff dout" (D54 at 111) and on April 27, 1995 Jeff Marr "[f]ixed mask used to enable for [inter]rupts - were not enabling the right bit." (*Id.* at 11). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D54.

68. Exhibit D55 shows the modifications for the week beginning on October, 1995. These modifications included approximately 170 changes to the Terpsichore system. (Exhibit D1). For example, on May 2, 1995 Richard Dickson "right justif[ied] rightmost section of [the integrated circuit] layout" (D55 at 52) and on May 4, 1995 Mark Semmelmeyer "fixed bug on nonblocking illegal loads for sych ops and bgate." (*Id.* at 2). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D55.

69. Exhibit D56 shows the modifications for the week beginning on May 6, 1995. These modifications included approximately 86 changes to the Terpsichore system. (Exhibit D1). For

example, on May 9, 1995 Jeff Marr "[t]ested cache thrashing with different LVA[63:48] and GVA[63:48] combinations" (D56 at 9) and on May 12, 1995, Geert Rosseel configured the Terpsichore systems "with new power assignments in the padring." (*Id.* at 1). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D56.

70. Exhibit D57 shows the modifications for the week beginning on May 13, 1995. These modifications included approximately 167 changes to the Terpsichore system. (Exhibit D1). For example, on May 16, 1995 Mark Semmelmeyer made "[m]inor corrections to I-cache & tag gate delays" and "[g]ave a write skew budget exploiting the 1 tick padding around the write enable to make the write skew not look like a violation." (D57 at 47). On May 17, 1995 Jay Tomlinson implemented a "[p]lacement change to help top-level routing, [r]educe congestion in center area … [and] [u]se up more of the empty space in the first 4 rows." (*Id.* at 73). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D57.

71. Exhibit D58 shows the modifications for the week beginning on May 20, 1995. These modifications included approximately 70 changes to the Terpsichore system. (Exhibit D1). For example, on May 26, 1995 Mark Semmelmeyer "[f]ixed optimistic error in rise/fall margin calculation" (D58 at 1) and on May 26, 1995 Jeff Marr "[f]ixed clear of hermes device event registers to use the uncached address." (*Id.* at 10). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D58.

72. Exhibit D59 shows the modifications for the week beginning on May 27, 1995. These modifications included approximately 106 changes to the Terpsichore system. (Exhibit D1). For example, on May 27, 1995 Tim Robinson "remove[d] force of power levels on cr interface to

prevent dc load problems" (D59at 5-6) and on May 27, 1995 Mark Semmelmeyer "[u]pdate[d] Cache and CTag address setup and hold times to use the newly-stick-approved 400ps requirements rather than the more a[r]bitrary 50%*tick requirement." (*Id.* at 34). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D59.

73. Exhibit D60 shows the modifications for the week beginning on June 3, 1995. These modifications included approximately 86 changes to the Terpsichore system. (Exhibit D1). For example, on June 4, 1995 Lisa Robinson "[r]elease[d] new dram macros" (D60 at 4) and on June 4, 1995 Warren Ong "[r]eplaced output with full swing FF." (*Id.* at 4). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D60.

74. Exhibit D61 shows the modifications for the week beginning on June 10, 1995. These modifications included approximately 99 changes to the Terpsichore system. (Exhibit D1). For example, on June 14, 1995 a "cycle correction adjustment" was performed (D61 at 11) along with a "new scheme for cycle correction." (*Id.* at 12). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D61.

75. Exhibit D62 shows the modifications for the week beginning on June 17, 1995. These modifications included approximately 69 changes to the Terpsichore system. (Exhibit D1). For example, on June 20, 1995 Jeff Marr "[t]est[ed] register dependencies and compare and swap" (D62 at 8) and on June 23, 1995, Tim Robinson "consolidate[d] routing congestion changes." (*Id.* at 18). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D62.

76. Exhibit D63 shows the modifications for the week beginning on June 24, 1995. These modifications included approximately 82 changes to the Terpsichore system. (Exhibit D1). For example, on June 28, 1995 Brian Smith "[a]dded simple event test for multiple events in a row" (D63 at 14) and on June 29, 1995 Jeff Marr "[t]est machine checks caused by illegal [serial bus] responses, and by parity errors in [serial bus] responses." (*Id.* at 15). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D63.

77. Exhibit D64 shows the modifications for the week beginning on July 1, 1995. These modifications included approximately 50 changes to the Terpsichore system. (Exhibit D1). For example, on July 3, 1995 Jay Tomlinson indicated that the Terpsichore system "passed multiple standalone tests" and indicated the tests passed (D64 at 9-10) and on July 4, 1995 Lisa Robinson "[h]ook[ed] hermes channel 1 up with dummy5 and dummy6." (*Id.* at 18). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D64.

78. Exhibit D65 shows the modifications for the week beginning on July 8, 1995. These modifications included approximately 81 changes to the Terpsichore system. (Exhibit D1). For example, on July 9, 1995 Lisa Robinson "[f]ixed an alignment error in thread 4" (D65 at 13) and on July 11, 1995 Lisa Robinson "[a]dd[ed] inverters for channel 1." (*Id.* at 22). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D65.

79. Exhibit D66 shows the modification for the week beginning on July 15, 1995. These modifications included approximately 92 changes to the Terpsichore system. (Exhibit D1). For example, on July 17, 1995 Derek Iverson "only let cyl 0 turn on dram and hermes channels"

(D66 at 21) and on July 20, 1995 Jeff Marr "[c]hange[d] max and min timer values – [Terpsichore system] is more accurate now, and caught that the test wasn't waiting long enough." (*Id.* at 22-23). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D66.

80. Exhibit D67 shows the modifications for the week beginning on July 22, 1995. These modifications included approximately 92 changes to the Terpsichore system. (Exhibit D1). For example, on July 26, 1995 Derek Iverson noticed problems with one of his previous changes and accordingly "modified [his testing] to introduce a dead-reckoned time delay for all cylinders but zero (if a delay is indeed required)" (D67 at 22), and on July 26, 1995, there was a "fix [to an] interrupt problem." (*Id.* at 13). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D67.

81. Exhibit D68 shows the modifications for the week beginning on July 29, 1995. These modifications included approximately 19 changes to the Terpsichore system. (Exhibit D1). For example, on July 26, 1995 Jeff Marr "[a]dded signals to trace hermes clocks." (D68 at 4). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D68.

82. Exhibit D69 shows the modifications for the week beginning on August 5, 1995. These modifications included approximately 71 changes to the Terpsichore system. (Exhibit D1). For example, on August 8, 1995 Mark Semmelmeyer fixed "[s]ome ICC and CC wires [that] were [unconnected]" (D69 at 29) and on August 10, 1995 Jay Tomlinson added information to the electronic database in order to allow "the [input-output] clock paths [to be tested]." (*Id.* at 26). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D69.

83. Exhibit D70 shows the modifications for the week beginning on August 12, 1995. These modifications included approximately 39 changes to the Terpsichore system. (Exhibit D1). For example, on August 13, 1995 Patricia Mayer "added 3 new resistors" (D70 at 7), and on August 16, 1995 Kurt Wampler "[a]dded notes on final Euterpe hand-routing strategy." (*Id.* at 1). Numerous other modifications were made to the Terpsichore system's electronic database during this week as shown in Exhibit D70.

Exhibit E – Payroll Records

84. Exhibit E reflects various MicroUnity payroll records covering the period from April 15, 1994 through August 15, 1995. Exhibit E1 is a summary of the total monthly head count from the three different departments at MicroUnity (Dept. Nos. 310, 320, and 450—which became department 350 in 1995) performing work on the Terpsichore system, and the bi-weekly expenditures associated with payroll for these individuals, from April 15, 1994 to August 15, 1995. Exhibit E2 contains copies of the actual pages of the Payroll Earnings Register for the aforementioned departments for the time period from April 15, 1994 to August 15, 1995 (the data from these pages was used to make the summary charts in Exhibit E1). Employee names and social security numbers have been redacted to maintain confidentiality. As can be appreciated from inspection of the summaries and their underlying documents, MicroUnity consistently spent between approximately $250,000 monthly on payroll for the design team, with a total expenditure of approximately $3M for the time period from October 1, 1994 until August 16, 1995 to develop the Terpsichore system.

85. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true, and the these statements are made with knowledge that willful false statements and the like so made are punishable by

fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, and any patent issuing thereon, or any patent to which this declaration is directed.

17-dept-2009
Date

Craig Hansen